LAMP-TR-004
CFAR-TR-849
CS-TR-3734

January 1997

Symbolic Compression and Processing of Document Images

Omid Kia, David Doermann, Azriel Rosenfeld, Rama Chellappa

Language and Media Processing Labratory
Instititue for Advanced Computer Studies
College Park, MD 20742

Abstract

In this paper we describe a compression and representation scheme which exploits the component-level redundancy found within a document image. The approach identifies patterns which appear repeatedly, represents similar patterns with a single prototype, stores the location of pattern instances and codes the residuals between the prototypes and the pattern instances. Using a novel encoding scheme, we provide a representation which facilitates scalable lossy compression and progressive transmission, and supports document image analysis in the compressed domain. We motivate the approach, provide details of the encoding procedures, report compression results and describe a class of document image understanding tasks which operate on the compressed representation.

***The support of the LAMP Technical Report Series and the partial support of this research by the National Science Foundation under grant EIA0130422 and the Department of Defense under contract MDA9049-C6-1250 is gratefully acknowledged.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE **JAN 1997**	2. REPORT TYPE	3. DATES COVERED **00-01-1997 to 00-01-1997**
4. TITLE AND SUBTITLE **Symbolic Compression and Processing of Document Images**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Language and Media Processing Laboratory,Institute for Advanced Computer Studies,University of Maryland,College Park,MD,20742-3275**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release; distribution unlimited**		
13. SUPPLEMENTARY NOTES		
14. ABSTRACT		
15. SUBJECT TERMS		

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**36**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

1 Introduction

Technological advances in processing, storage, and visualization have made it possible to maintain large numbers of documents in digital image form and make them accessible over networks. In order to do this effectively, three primary concerns must be addressed. The first is document size. An ASCII version of a document page can easily be stored in 2-3 KB, whereas a typical scanned page may result in an image which requires between 500 KB and 2 MB. A single book stored in image form can fill a CD-ROM to capacity. If we are to maintain documents in image form, an efficient compression scheme is essential for both storage and transmission.

The second concern is that of efficient access to the compressed images. Traditional compression techniques used for document images have been successful in reducing storage requirements but do not provide efficient access to the compressed data. It is desirable to use a compression method that makes use of a structured representation of the data, so that it not only allows for rapid transmission but also promotes compressed-domain processing and allows access to various document components in the compressed domain.

The third concern is that of readability. Most lossy compression and progressive transmission techniques use resolution reduction or texture-preserving methods that might render a document image unreadable. It is desirable that a document be readable even at the highest levels of lossy compression and at the start of a progressive transmission; it can then be augmented by subsequent information for better rendition. This is much preferable to a scenario in which the highest resolution is the only readable resolution.

2 Background

Before describing our approach, it is useful to briefly review the general problem of image compression, issues in document processing which affect compression, and document image compression itself.

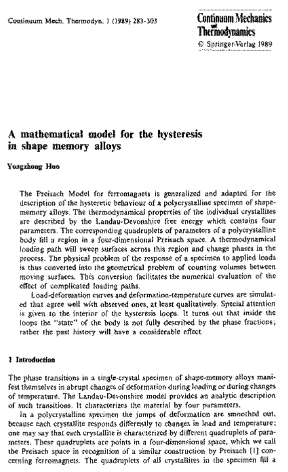
2.1 General compression issues

Compression techniques offer an attractive option for storing large amounts of data efficiently. Most approaches utilize signal compression techniques to reduce redundancy by transforming the source data to a stream which represents the data by a compact encoding. The characteristics of the transformation allow various degrees of compressibility; commonly used methods include techniques such as vector quantization [10] and transform coding [17], among others. After appropriate encoding has been performed, techniques such as Shannon-Fano coding [38], Huffman coding [8, 9, 23, 42, 47, 48], and run-length coding [14] can be used to represent the encoded data efficiently. The transformation can be tuned for specific media characteristics, and in the case of lossy compression, can be tuned to preserve specific properties of the source data. Many lossy compression methods use pixel-level redundancy and perform resolution reduction or texture-preserving operations to achieve compression. Such approaches are unsuitable for document images since much of the usable information is found at the symbol level rather than at the pixel level or in the texture. While preserving the overall appearance of the document, resolution reduction may render the text portions unreadable; and any attempt to texturally encode a document image will fail since the useful information in such an image takes the form of high-contrast, high-gradient, locally non-uniform pixel patterns. When we perform lossy compression, this must be done in such a way as to eliminate only data which will not affect the usability (readability) of the document.

2.2 Document characteristics

Document images are scans of documents which are in most cases pseudo-binary and rich in textual content. Informally, we can define document images as images that contain components that resemble the symbols of a language. Many documents look like those shown in Figure 1 and are represented adequately by binary images produced by scanning at a resolution of 300 dots per inch (dpi). They tend to be highly structured in terms of layout, and have significant redundancy in the symbols which occur in them.

Since a document can be used in a large number of ways, an important consideration is how the image is affected by compression. For example, if we intend that the document ultimately be read by humans, it is necessary for compression schemes to preserve the shapes of the components so that they are recognizable by the reader after retrieval. If the document must be reproduced in near-original form, techniques which reduce resolution may not be acceptable. Thus the required resolution is dependent on the task; some resolutions may leave the document readable, but may destroy fine detail.



Continuum Mech. Thermodyn. 1 (1989) 283-303

Continuum Mechanics and Thermodynamics
© Springer-Verlag 1989

A mathematical model for the hysteresis in shape memory alloys

Yongzhong Huo

The Preisach Model for ferromagnets is generalized and adapted for the description of the hysteretic behaviour of a polycrystalline specimen of shape-memory alloys. The thermodynamical properties of the individual crystallites are described by the Landau-Devonshire free energy which contains four parameters. The corresponding quadruplets of parameters of a polycrystalline body fill a region in a four-dimensional Preisach space. A thermodynamical loading path will sweep surfaces across this region and change phases in the process. The physical problem of the response of a specimen to applied loads is thus converted into the geometrical problem of counting volumes between moving surfaces. This conversion facilitates the numerical evaluation of the effect of complicated loading paths.

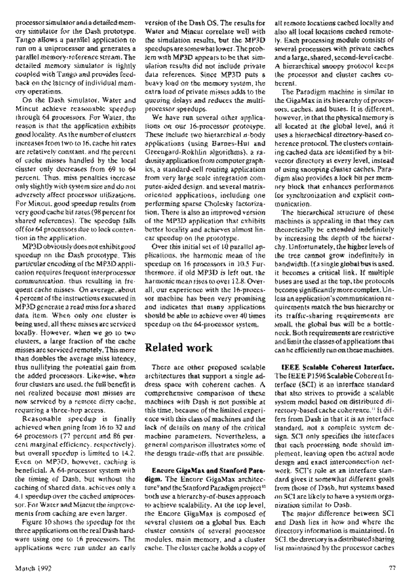
Load-deformation curves and deformation-temperature curves are simulated that agree well with observed ones, at least qualitatively. Special attention is given to the interior of the hysteresis loops. It turns out that inside the loops the "state" of the body is not fully described by the phase fractions; rather the past history will have a considerable effect.

1 Introduction

The phase transitions in a single-crystal specimen of shape-memory alloys manifest themselves in abrupt changes of deformation during loading or during changes of temperature. The Landau-Devonshire model provides an analytic description of such transitions. It characterizes the material by four parameters.

In a polycrystalline specimen the jumps of deformation are smoothed out, because each crystallite responds differently to changes in load and temperature; one may say that each crystallite is characterized by different quadruplets of parameters. These quadruplets are points in a four-dimensional space, which we call the Preisach space in recognition of a similar construction by Preisach [1] concerning ferromagnets. The quadruplets of all crystallites in the specimen fill a

(a)



processor simulator and a detailed memory simulator for the Dash prototype. Tango allows a parallel application to run on a uniprocessor and generates a parallel memory-reference stream. The detailed memory simulator is tightly coupled with Tango and provides feedback on the latency of individual memory operations.

On the Dash simulator, Water and Mincut achieve reasonable speedup through 64 processors. For Water, the reason is that the application exhibits good locality. As the number of clusters increases from two to 16, cache hit rates are relatively constant, and the percent of cache misses handled by the local cluster only decreases from 69 to 64 percent. Thus, miss penalties increase only slightly with system size and do not adversely affect processor utilizations. For Mincut, good speedup results from very good cache hit rates (98 percent for shared references). The speedup falls off for 64 processors due to lock contention in the application.

MP3D obviously does not exhibit good speedup on the Dash prototype. This particular encoding of the MP3D application requires frequent interprocessor communication, thus resulting in frequent cache misses. On average, about 4 percent of the instructions executed in MP3D generate a read miss for a shared data item. When only one cluster is being used, all these misses are serviced locally. However, when we go to two clusters, a large fraction of the cache misses are serviced remotely. This more than doubles the average miss latency, thus nullifying the potential gain from the added processors. Likewise, when four clusters are used, the full benefit is not realized because most misses are now serviced by a remote dirty cache, requiring a three-hop access.

Reasonable speedup is finally achieved when going from 16 to 32 and 64 processors (77 percent and 86 percent marginal efficiency, respectively), but overall speedup is limited to 14.2. Even on MP3D, however, caching is beneficial. A 64-processor system with the timing of Dash, but without the caching of shared data, achieves only a 4.1 speedup over the cached uniprocessor. For Water and Mincut the improvements from caching are even larger.

Figure 10 shows the speedup for the three applications on the real Dash hardware using one to 16 processors. The applications were run under an early version of the Dash OS. The results for Water and Mincut correlate well with the simulation results, but the MP3D speedups are somewhat lower. The problem with MP3D appears to be that simulation results did not include private data references. Since MP3D puts a heavy load on the memory system, the extra load of private misses adds to the queuing delays and reduces the multiprocessor speedups.

We have run several other applications on our 16-processor prototype. These include two hierarchical *n*-body applications (using Barnes-Hut and Greengard-Rokhlin algorithms), a radiosity application from computer graphics, a standard-cell routing application from very large scale integration computer-aided design, and several matrix-oriented applications, including one performing sparse Cholesky factorization. There is also an improved version of the MP3D application that exhibits better locality and achieves almost linear speedup on the prototype.

Over this initial set of 10 parallel applications, the harmonic mean of the speedup on 16 processors in 10.5 Furthermore, if old MP3D is left out, the harmonic mean rises to over 12.8. Overall, our experience with the 16-processor machine has been very promising and indicates that many applications should be able to achieve over 40 times speedup on the 64-processor system.

Related work

There are other proposed scalable architectures that support a single address space with coherent caches. A comprehensive comparison of these machines with Dash is not possible at this time, because of the limited experience with this class of machines and the lack of details on many of the critical machine parameters. Nevertheless, a general comparison illustrates some of the design trade-offs that are possible.

Encore GigaMax and Stanford Paradigm. The Encore GigaMax architecture[9] and the Stanford Paradigm project[10] both use a hierarchy-of-buses approach to achieve scalability. At the top level, the Encore GigaMax is composed of several clusters on a global bus. Each cluster consists of several processor modules, main memory, and a cluster cache. The cluster cache holds a copy of all remote locations cached locally and also all local locations cached remotely. Each processing module consists of several processors with private caches and a large, shared, second-level cache. A hierarchical snoopy protocol keeps the processor and cluster caches coherent.

The Paradigm machine is similar to the GigaMax in its hierarchy of processors, caches, and buses. It is different, however, in that the physical memory is all located at the global level, and it uses a hierarchical directory-based coherence protocol. The clusters containing cached data are identified by a bit-vector directory at every level, instead of using snooping cluster caches. Paradigm also provides a lock bit per memory block that enhances performance for synchronization and explicit communication.

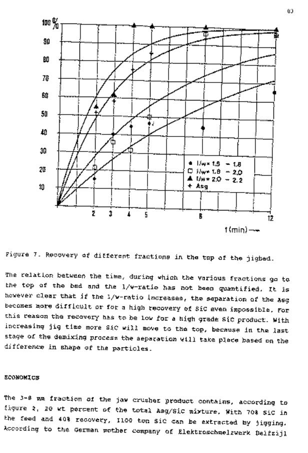
The hierarchical structure of these machines is appealing in that they can theoretically be extended indefinitely by increasing the depth of the hierarchy. Unfortunately, the higher levels of the tree cannot grow indefinitely in bandwidth. If a single global bus is used, it becomes a critical link. If multiple buses are used at the top, the protocols become significantly more complex. Unless an application's communication requirements match the bus hierarchy or its traffic-sharing requirements are small, the global bus will be a bottleneck. Both requirements are restrictive and limit the classes of applications that can be efficiently run on these machines.

IEEE Scalable Coherent Interface. The IEEE P1596 Scalable Coherent Interface (SCI) is an interface standard that also strives to provide a scalable system model based on distributed directory-based cache coherence.[11] It differs from Dash in that it is an interface standard, not a complete system design. SCI only specifies the interfaces that each processing node should implement, leaving open the actual node design and exact interconnection network. SCI's role as an interface standard gives it somewhat different goals from those of Dash, but systems based on SCI are likely to have a system organization similar to Dash.

The major difference between SCI and Dash lies in how and where the directory information is maintained. In SCI, the directory is a distributed sharing list maintained by the processor caches

March 1992 77

(b)



100 %
90
80
70
60
50
40
30
20
10
2
3
4
5
8
12
t(min) →
● l/w= 1.5 – 1.8
□ l/w= 1.8 – 2.0
▲ l/w= 2.0 – 2.2
+ Asg

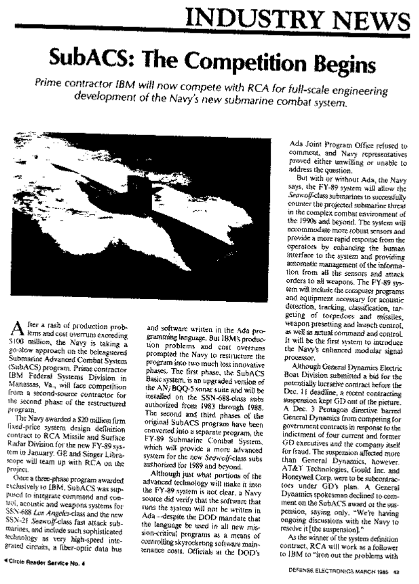
Figure 7. Recovery of different fractions in the top of the jigbed.

The relation between the time, during which the various fractions go to the top of the bed and the l/w-ratio has not been quantified. It is however clear that if the l/w-ratio increases, the separation of the Asg becomes more difficult or for a high recovery of SiC even impossible. For this reason the recovery has to be low for a high grade SiC product. With increasing jig time more SiC will move to the top, because in the last stage of the demixing process the separation will take place based on the difference in shape of the particles.

ECONOMICS

The 3-8 mm fraction of the jaw crusher product contains, according to figure 2, 20 wt percent of the total Asg/SiC mixture. With 70% SiC in the feed and 40% recovery, 1100 ton SiC can be extracted by jigging. According to the German mother company of Elektroschmelzwerk Delfzijl

(c)



INDUSTRY NEWS

SubACS: The Competition Begins

Prime contractor IBM will now compete with RCA for full-scale engineering development of the Navy's new submarine combat system.

After a rash of production problems and cost overruns exceeding $100 million, the Navy is taking a go-slow approach on the beleaguered Submarine Advanced Combat System (SubACS) program. Prime contractor IBM Federal Systems Division in Manassas, Va., will face competition from a second-source contractor for the second phase of the restructured program.

The Navy awarded a $20 million firm fixed-price system design definition contract to RCA Missile and Surface Radar Division for the new FY-89 system in January. GE and Singer Librascope will team up with RCA on the project.

Once a three-phase program awarded exclusively to IBM, SubACS was supposed to integrate command and control, acoustic and weapons systems for SSN-688 *Los Angeles*-class and the new SSN-21 *Seawolf*-class fast attack submarines, and include such sophisticated technology as very high-speed integrated circuits, a fiber-optic data bus and software written in the Ada programming language. But IBM's production problems and cost overruns prompted the Navy to restructure the program into two much less innovative phases. The first phase, the SubACS Basic system, is an upgraded version of the AN/BQQ-5 sonar suite and will be installed on the SSN-688-class subs authorized from 1983 through 1988. The second and third phases of the original SubACS program have been converted into a separate program, the FY-89 Submarine Combat System, which will provide a more advanced system for the new *Seawolf*-class subs authorized for 1989 and beyond.

Although just what portions of the advanced technology will make it into the FY-89 system is not clear, a Navy source did verify that the software that runs the system will not be written in Ada—despite the DOD mandate that the language be used in all new mission-critical programs as a means of controlling skyrocketing software maintenance costs. Officials at the DOD's Ada Joint Program Office refused to comment, and Navy representatives proved either unwilling or unable to address the question.

But with or without Ada, the Navy says, the FY-89 system will allow the *Seawolf*-class submarines to successfully counter the projected submarine threat in the complex combat environment of the 1990s and beyond. The system will accommodate more robust sensors and provide a more rapid response from the operators by enhancing the human interface to the system and providing automatic management of the information from all the sensors and attack orders to all weapons. The FY-89 system will include the computer programs and equipment necessary for acoustic detection, tracking, classification, targeting of torpedoes and missiles, weapon presetting and launch control, as well as actual command and control. It will be the first system to introduce the Navy's enhanced modular signal processor.

Although General Dynamics Electric Boat Division submitted a bid for the potentially lucrative contract before the Dec. 11 deadline, a recent contracting suspension kept GD out of the picture. A Dec. 3 Pentagon directive barred General Dynamics from competing for government contracts in response to the indictment of four current and former GD executives and the company itself for fraud. The suspension affected more than General Dynamics, however. AT&T Technologies, Gould Inc. and Honeywell Corp. were to be subcontractors under GD's plan. A General Dynamics spokesman declined to comment on the SubACS award or the suspension, saying only, "We're having ongoing discussions with the Navy to resolve it [the suspension]."

As the winner of the system definition contract, RCA will work as a follower to IBM to "iron out the problems with

◄Circle Reader Service No. 4

DEFENSE ELECTRONICS MARCH 1986 43

(d)

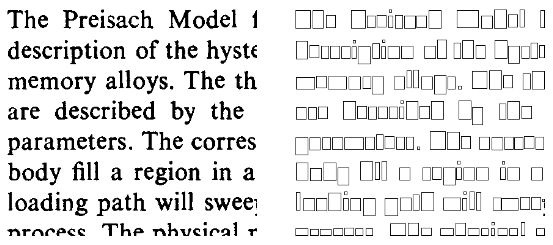
Figure 1: Examples of binary document images of the types usually found in a document image database. These are images from the University of Washington document database [11], cropped (automatically) to the main body of the text.

As is well known, the performance of conventional compression algorithms (such as those based on transform coding [17], vector quantization [10], fractal compression [5], pattern matching and substitution [3, 25], and other approaches) depends on the types of images being compressed, and on their texture and content characteristics. Algorithms often do well on some classes of images and not so well on others. Since document images differ significantly from scene images, it is reasonable to assume they will benefit from a specialized compression scheme.

2.3 Document compression

Document image compression was first recognized as an important problem in the 1970s when increased use of facsimile machines led to growth in document scanning and transmission. The international telephone and telegraph communication committee (CCITT) then set out to create a standard for the use of compression in facsimile transmission. In 1980 the first comprehensive standard was released as the CCITT Group 3 transmission standard. This was followed by CCITT Group 4 [7, 36, 12], which was originally meant for digital, as opposed to analog, transmission. Both schemes use a run-length-based approach, with an extension to dual scan-line coding to exploit coherence between successive scan lines. In Group 4, higher compression is achieved due largely to the omission of error-correcting codes which were intentionally included in the Group 3 standard. The Joint Binary Image Group (JBIG) standard is a recent international CCITT standard in which context modeling forms the basis of coding [33, 34]. At the lowest levels JBIG uses a template model and adaptive arithmetic coding to encode predictions. Layers of the image can be encoded separately to provide progressive transmission, based on resolution reduction and the use of exception pixels. More recently, new developments in vector quantization, finite automata theory [15], and pattern-based approaches [13] have provided alternative methods for document compression with the ability to perform lossy compression. These techniques, however, were not intended to facilitate compressed-domain processing, and very few of them show promise for future extensions in that direction.

A basic approach to document compression, first suggested by Ascher and Nagy [2] and later formalized by Witten et al. [45, 16, 44, 43], attempts to encode redundancy at the symbol level. Ascher and Nagy proposed using a dynamic library in which document image components are extracted and repeated components are indexed to the first instance of each component. Witten


The Preisach Model
description of the hyste
memory alloys. The th
are described by the
parameters. The corres
body fill a region in a
loading path will swee
process. The physical

(a) (b)

Figure 2: a) A sample document image; b) the bounding boxes of its connected components.

et al. proposed an enhancement of this approach by encoding an image in terms of constituent patterns and residuals. This method is very similar to many pattern matching and substitution algorithms [3, 18, 25, 27, 28, 36, 46], and provides a robust way to determine the similarity between the components in an image. Redundancies in symbol shape are coded, resulting in redundancy reduction in the two-dimensional pattern space as opposed to the traditional reduction in the one-dimension pixel stream. Refinements and extensions of this basic concept, such as those described in this paper, can provide a powerful basis for developing an approach to compression for document image management systems.

2.4 Approach

The first step in our symbolic document image compression method is to find an initial set of patterns in the image which can be used to form a library. In the case of Latin text, performing a connected component analysis on the binary image provides a reasonable starting set. For connected scripts or text in which the basic units are disconnected, more extensive segmentation would be necessary.

A small portion of a typical document is shown in Figure 2a, and the bounding boxes of the connected components are shown in Figure 2b. Because these patterns tend to appear repeatedly in the image, they form a basis for compression. In cases where multiple characters touch to form a single component, or where a single character is split into multiple components, representing them as a new component lowers the compression factor only slightly. If salt and pepper noise is present, it may give rise to small components; this will reduce compression but will have little or no effect on the readability of the document.

We next treat each component as an observation and try to determine a best set of classes (clusters) of the components and to choose a prototype image for each class. We begin by comparing each observed component to all previously chosen prototypes (Figure 3). If a match exists, we assign the observation to that class and refine the prototype of the class. If no match is sufficiently close we regard the observation as defining a new class and take the observation as a prototype for that class. After all observations have been processed, the prototype map typically looks like the one shown in Figure 4. The shapes shown in Figure 4 resemble English characters because of the primarily English content of the original document. For a document rich in mathematical symbols, some of the prototypes would resemble mathematical symbols, and similarly for non-Latin languages the prototypes would capture their symbol content. A number of matching algorithms can be used to group similar patterns into clusters, including simple XOR, weighted XOR, Boolean AND-NOT, and compression-based template matching. They will be discussed in Section 3.

Each cluster of components is represented by a prototype. Depending on the sample space, there will exist some amount of variability in the clusters. For some clusters, the amount of this variation may be large enough that some of the components differ significantly from the prototype and extra information must be recorded to remedy this. A residual map, the difference between a given component and its prototype, is preserved and used to recover the components in a lossless form when necessary. Examples of a component, a prototype, and the corresponding residual map are shown in Figure 5. In addition to coding the prototype, we code the residual map separately so that access to individual symbols can be achieved by access to their residual maps. The overall encoding scheme is shown in Figure 6.

In our work special attention is given to performing document analysis tasks in the compressed domain, without full decompression. In many situations only parts of the encoded information pertaining to the given task require decompression.

Our approach makes a strong case for the use of symbolic compression in document image coding. We will show that it is possible to code the dominant symbol shapes and their locations within the image using only about one percent of the original image data. Using only this encoding it is possible to implement a large number of common document analysis tasks such as skew estimation/correction [24, 30, 35, 6, 4], Optical Character Recognition (OCR) [29], and layout analysis [31, 32].

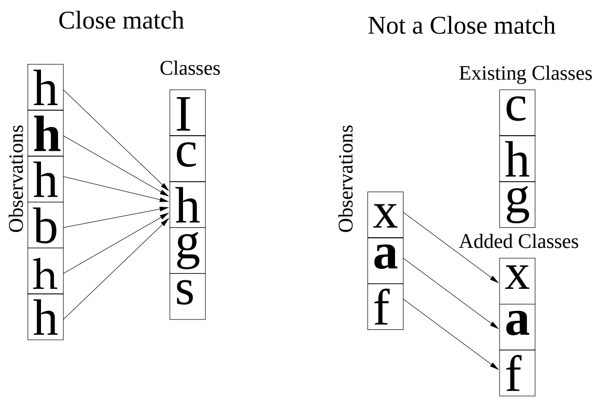


Figure 3: Clustering by pattern matching

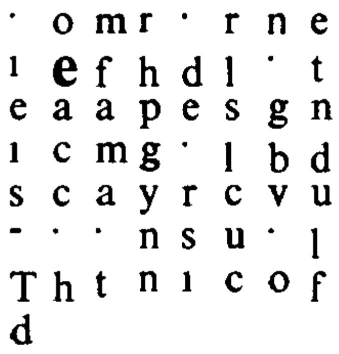
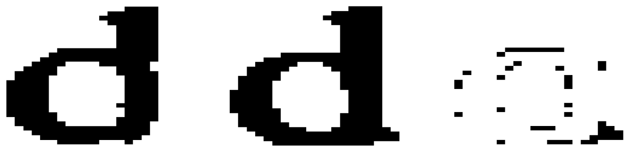


Figure 4: Typical cluster prototypes from a textually rich image



(a) (b) (c)

Figure 5: a) A component; b) a prototype; c) the residual map.

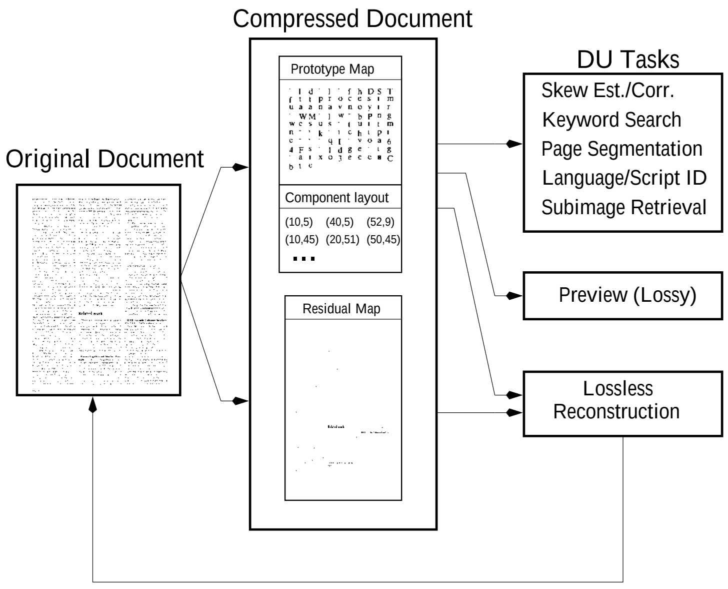


Figure 6: Representation of a compressed document image with relevant processing access.

Although some work has been done on the processing of compressed document images [26, 40, 41] the outlook for performing such processing on standard pixel-based compression representations does not appear promising. In our approach, we use an indexable representation [22] composed of independent streams for the prototypes, the locations of symbol instances, and the residual maps. We have shown that lossy compression, progressive transmission, sub-document retrieval, skew estimation/correction, and keyword searching can all be done efficiently using this representation [22].

The main contribution of this paper is the development of a compression system that promotes compressed-domain analysis by allowing symbol access. Although the approach works best with clean images where multiple patterns repeat, it is flexible enough to adapt to situations where the components correspond to arbitrary patterns in the image as opposed to symbols, or where many symbols are mis-clustered.

The remainder of this paper is organized as follows. Section 3 describes the algorithms to generate clusters. Section 4 describes the approach used to represent the clusters and Section 5 discusses the approach to page representation and the properties it preserves. Section 6 briefly covers aspects of the implementation. A detailed discussion of applications that benefit from this

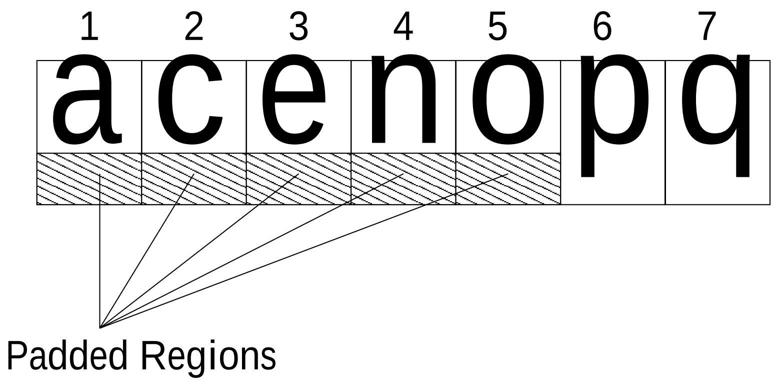


Figure 7: Example of padding

implementation is given in Section 7, and Section 8 provides a summary of the overall system and discusses directions for further research.

3 Clustering

A large number of clustering algorithms have been described in the literature [1, 37]. Some algorithms, despite their wide usage for other purposes, are not applicable to symbol clustering. The goal of symbol clustering is to identify the intra-class similarities of the symbols.

One example of an inappropriate clustering algorithm is principal component analysis, where image rows or columns are concatenated into a vector and for a given set of symbol images a covariance matrix is computed. The eigenvalues of this matrix give the variances along the eigenvector directions. The effectiveness of this method depends on the rows and columns providing consistent information for all of the images. If the images are of different sizes, this will not be the case. To remedy this one can pad the observed image to a fixed size with a constant value. This padding presents another problem by giving weights to the image values in the padded regions. In the example shown in Figure 7, it can be seen that all observations have a similar shape in the upper portion of the image, and that the lower portions of observations 6 and 7 look the most dissimilar. However, since observations 1 to 5 are padded, the weight of the descenders in observations 6 and 7 is 2/7 of what it should be. As a matter of fact there should be a discrimination between observations $\{6,7\}$ and observations $\{1,2,3,4,5\}$ because of their size.

A second type of clustering algorithm, vector quantization, is also affected by the variability in observed sizes. In vector quantization, each symbol image is first ordered as a vector and this vector is considered to be a point in a high-dimensional space. A small set of "prototype" vectors is arbitrarily chosen and the distances to all prototypes from all of the observations are calculated. We associate with each observation the prototype closest to it. The prototypes are then refined by taking the means of their associated observations, and the process is repeated. The prototypes and the distance measure divide the high-dimensional space into regions of coarser granularity; hence the name quantization. The results obtained by this method are population-dependent; more prototypes will be found in regions of the space where symbol images occur. Figure 8 shows how the regions change when the number of samples in one of the clusters is increased.

A third class of clustering methods, which we will primarily use here, are pattern matching and substitution approaches. If a candidate pattern is a good match to an existing prototype, it is classified as a member of that prototype's class; otherwise, it is considered for possible creation of a new class. The advantages of using this type of method are that we do not need a priori knowledge about the number of classes and that the method works with a variety of image sizes.

We can describe clustering via pattern matching more formally by first defining notation for an observation X and a prototype P:

$$X = \begin{cases} X(n,m) & \text{for } (n,m) \in \mathcal{R}_x = (\{1,...,N_x\},\{1,...,M_x\}) \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$$P = \begin{cases} P(n,m) & \text{for } (n,m) \in \mathcal{R}_p = (\{1,...,N_p\},\{1,...,M_p\}) \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where it is assumed that foreground pixels have value 1 and background pixels (and pixels outside the bounds of $\mathcal{R}_x$ and $\mathcal{R}_p$) have value 0. A number of pattern matching methods which can be used to define measures of closeness between an observation and a prototype are discussed below.

3.1 Hamming distance

The simplest method of matching two binary images is to measure their dissimilarity by the number of pixels that are not equal. An error map calculated from the exclusive OR (XOR) of the observed

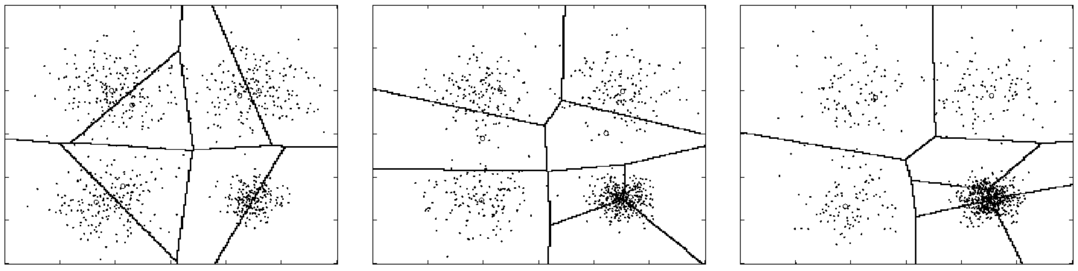


Figure 8: Vector quantization regions in a two-dimensional sample space with a) equal-population clusters; b) a 3:1 population difference; c) a 5:1 population difference.

image and the prototype is given by

$$E(n,m) = \begin{cases} X(n,m) \oplus P(n,m) & \text{for } (n,m) \in \mathcal{R}_x \cap \mathcal{R}_p \\ X(n,m) & \text{for } (n,m) \in \mathcal{R}_x - \mathcal{R}_p \\ P(n,m) & \text{for } (n,m) \in \mathcal{R}_p - \mathcal{R}_x \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

where $E(n,m)$ is defined for all $(n,m) \in \mathcal{R}_e = \mathcal{R}_x \cup \mathcal{R}_p$. Since the XOR operation returns a value of 1 for a mismatch, summing the error map provides a measure of mismatch,

$$\overline{M} = |\mathcal{R}_e| - M = \sum_{(n,m)\in\mathcal{R}_e} E(n,m) \tag{4}$$

where $|\mathcal{R}_e|$ is the highest mismatch score and M is the actual match. In this formulation, maximizing the match is the same as minimizing the mismatch of equation (4). Figure 9 shows three examples involving an intra-class observation, a possibly confusing inter-class observation, and an obvious inter-class observation, along with their mismatch scores. The amount of mismatch shows some degree of discrimination among those cases, and by use of a threshold, we are able to identify some classes. However, if the threshold is too high some intra-class confusion may arise, such as clustering 'e' shapes with 'c' shapes. Taking a low threshold would result in too many clusters and would hinder compression efforts. It is desirable to use a distance measure that provides enough separation for dissimilar shapes.

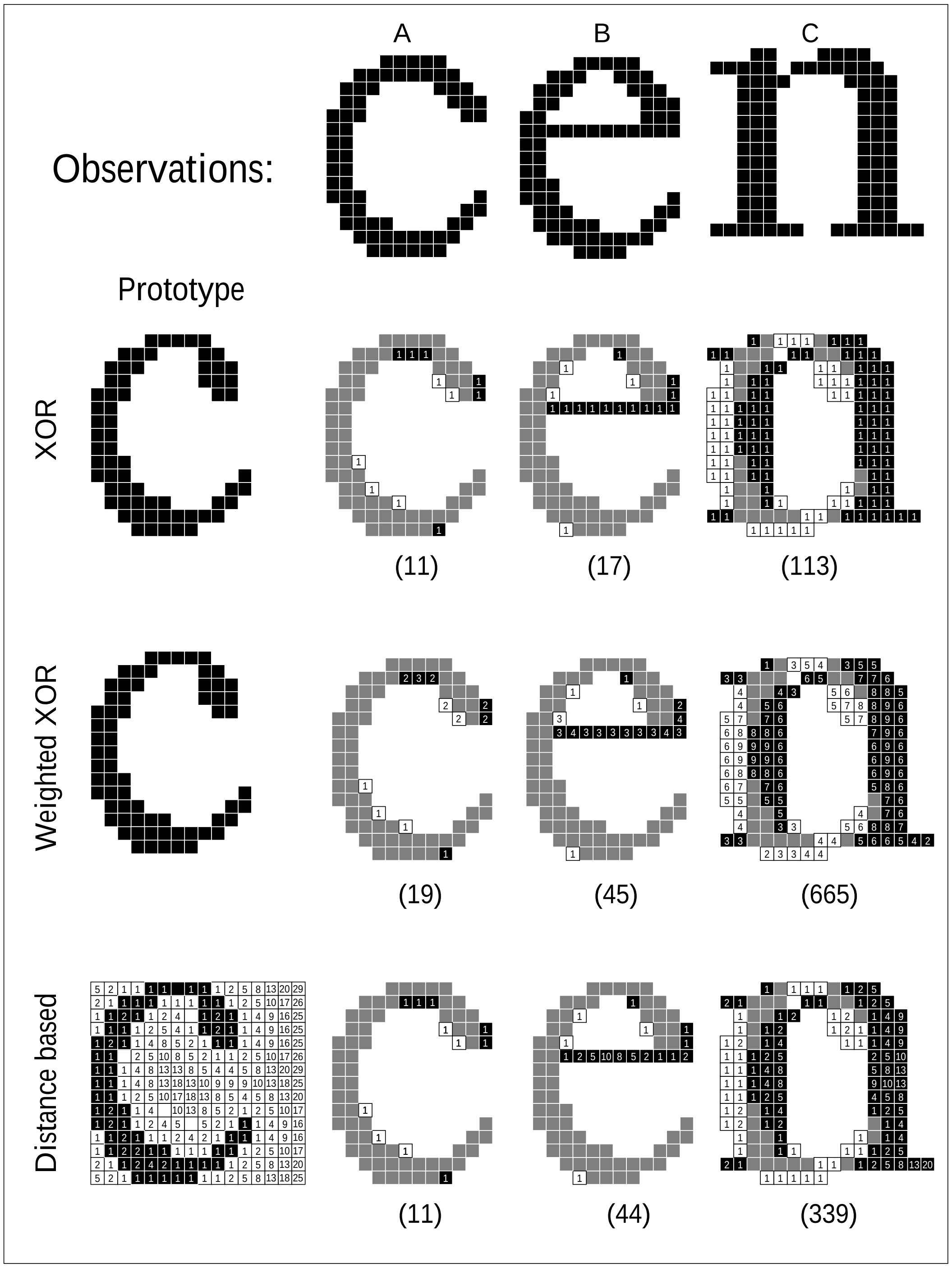


Figure 9: Matching results for three examples of observed components. Observation A is similar to the prototype, observation B is close, and observation C is very different. XOR-ed, Weighted XORed, and Distance-based differences are shown, as well as the pixels contributing to the mismatch, and the mismatch value in parentheses.

3.2 Weighted Hamming distance

Improving the distance measure to discriminate between similar images ('e' and 'c' of Figure 9) is desirable; the weighted Hamming distance [36, 45] provides such an improvement. This distance measure gives greater importance to error pixels which appear in close proximity to other error pixels. Error pixels which appear close together tend to correspond to structurally meaningful features.

The weighted Hamming distance operates on the error map of equation (3) by summing over a neighborhood of each error pixel:

$$E_w(n,m) = E(n,m) \times \sum_{(k,l)\in\mathcal{N}(n,m)} E(k,l) \tag{5}$$

where $\mathcal{N}(n,m)$ is the 3×3 neighborhood of the (n,m)th pixel. With this weighting strategy, error pixels that occur in a group will give a higher mismatch than isolated error pixels. Figure 9 also shows the results of using the weighted Hamming distance measure for the same observations and prototype. Note that the small difference between observations A and B is now much larger when the mismatch score is calculated by equation (4) and using the weighted error map E_w.

3.3 Sum of weighted AND-NOTs

When we use an XOR operation the source of the errors is not considered. In particular no distinction is made between errors in foreground pixels and errors in background pixels. It may be desirable to give more importance to the foreground pixels since most of the information is contained in them. This can be done by using the AND-NOT measure. The weighted AND-NOT map is defined by

$$\begin{aligned} E_{wan} &= \left[(X \wedge \overline{P})(n,m) \times \sum_{(k,l)\in\mathcal{N}(n,m)} (X \wedge \overline{P})(k,l)\right] \\ &\vee \left[(\overline{X} \wedge P)(n,m) \times \sum_{(k,l)\in\mathcal{N}(n,m)} (\overline{X} \wedge P)(k,l)\right] \end{aligned} \tag{6}$$

This map is useful in cases where the weighting has elevated the mismatch level due to misalignment, as illustrated in Figure 10.

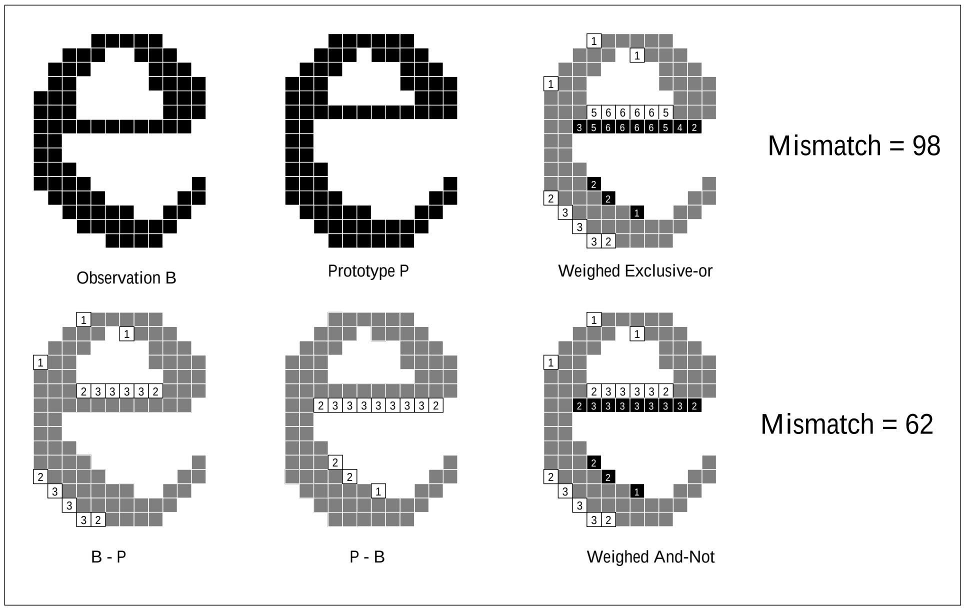


Figure 10: Example of an observed component compared to a prototype by measuring mismatch as weighted XOR and as weighted AND-NOT.

3.4 Compression-based template matching

Compression-based template matching [16] is a method of matching which attempts to measure the mutual information between a prototype and an observation. This is done by measuring the entropy of the residual left after matching the observation with the prototype, and trying to minimize it. The entropy of a binary signal E which is composed of samples e_i for $i = 1, ..., |\mathcal{R}_e|$ is defined as

$$H(E) = -\Pr[e = 0]\log(\Pr[e = 0]) - \Pr[e = 1]\log(\Pr[e = 1]) \quad (7)$$

$\Pr[e = 0]$, the probability of a background sample, is defined as the number of 0 samples divided by the total number of samples, and $\Pr[e = 1]$, the probability of a foreground sample, is then $1 - \Pr[e = 0]$. A plot of the entropy as a function of $\Pr[e = 0]$ is shown in Figure 11. Note that the maximum entropy is reached at $\Pr[e = 0] = 0.5$ and the minimum at $\Pr[e = 0] = 0$ or 1 (i.e. an all-0 or all-1 residual map). This is rather different from previous approaches in which the number of pixels in the residual map is minimized. In compression-based template matching, the idea is to keep the residual pixels non-random with respect to each other and achieve a lower entropy for the residual map. This is also beneficial for coding since the residual is supposed to be coded, and a

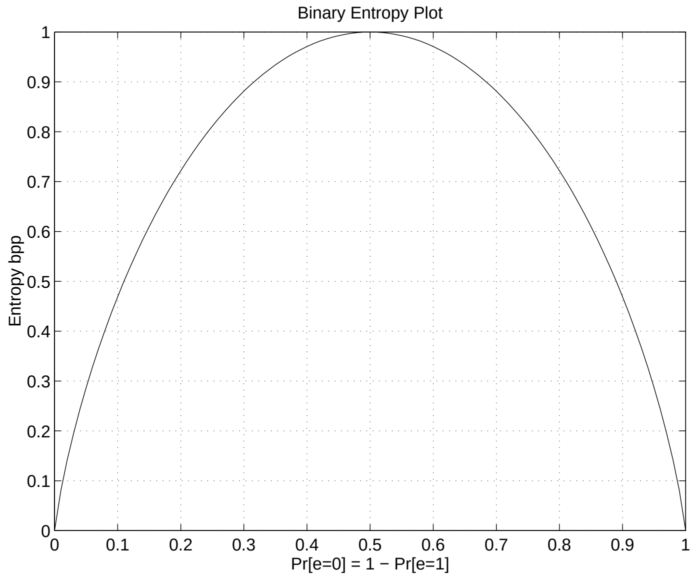


Figure 11: Plot of entropy for a binary signal as a function of sample probability

lower entropy translates into a higher compression factor.

3.5 Distance-based template matching

Our final method of matching attempts to weight each error pixel according to the probability that it was corrupted by degradation or noise. In this way we give lower weights to error pixels which are likely to have resulted from noise and higher weights to others.

In studies of document image degradation at the character level it was observed that pixel errors occur more often close to the edges of characters [19]. The probability of a pixel changing its value decays as a squared exponential with the distance to an edge. Specifically [19], the probability of a pixel changing value from background to foreground at distance d from an edge is given by

$$P(1|d,\alpha,b) = 1 - P(0|d,\alpha,b) = e^{-\alpha d^2} \tag{8}$$

where α is the background decay rate and b denotes the fact that the pixel in question really belongs to the background. Similarly, for foreground pixels, the probability of changing value is

$$P(0|d,\beta,f) = 1 - P(1|d,\beta,f) = e^{-\beta d^2} \tag{9}$$

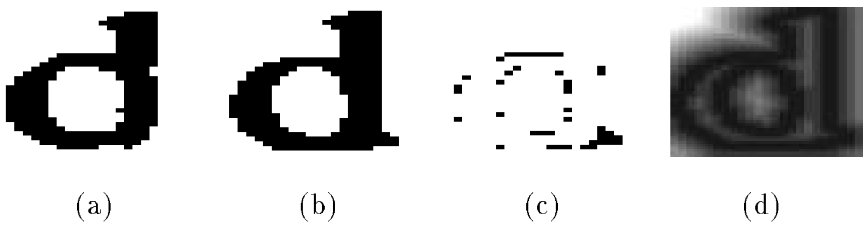


Figure 12: Example of a) an observation, b) a prototype, c) a residual map, and d) the distance map of the prototype.

We can use this fact to conclude that observed differences which occur between a prototype and a candidate pattern away from the edge of the prototype are more likely to be the result of meaningful structure and not noise and should therefore be considered with higher priority during matching.

We can use this degradation model as a basis for a matching model which assigns weights based on relative distances from the prototype edge. To simplify the model, we assume that α is equal to β and calculate the weight matrix as a distance transform. This greatly reduces the computational requirement by avoiding a scan to all components to determine α and β and by removing an extra exponential operation. We also use only a relative distance measure; this allows us to avoid the computation of two squares and a square root.

Figure 12 shows a typical prototype and a map of the distances to the edges. (Figure 9 showed the distances overlayed on the prototype image.) Summing the error pixels weighted by their distance values provides a better separation for inter-class observations. For the intra-class observations in Figure 10, it is easy to see that the distance-weighted mismatch is 26 since all error pixels occur on the edges and there are only 26 such pixels. Distance weighting separates the closest inter-class observations shown in Figure 9 by 11 and 44.

It should be pointed out that distance-based template matching could easily be adapted to other clustering techniques, at the cost of more computation. An example is adaptation to vector quantization. Consider the clusterings of Figure 8, and suppose that for high values of x and y we would like to weight the x values more heavily, and for low values we would like to weight the y values more heavily. This would shift the region boundaries of Figure 8a in such a way that at the top right we would have vertical thin regions and at the lower left we would have horizontal thin regions. The result would be a partition that is dependent on location in the space as well as on the population of observations in that area of the space. The distance ordering has a number of

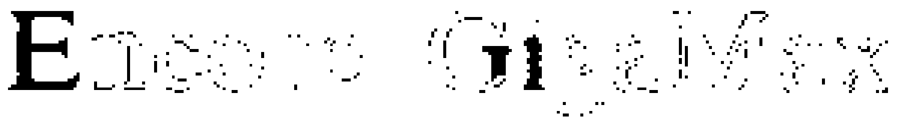


Figure 13: Example of ambiguous membership and resulting residual.

desirable properties, and provides a good framework for document image processing and handling.

4 Cluster representation

Once we have obtained clusters and chosen prototypes, two concerns must be addressed: representing the residuals (the differences between the components and their associated prototypes) and specifying the positions of the components in the image.

4.1 Residual representation

Replacement of the observations by the prototypes, which are the centers of clusters of observations, yields a representation that does not reduce spatial resolution. The replacement may give rise to three types of residuals. The first type is the most common variation from the prototype; it takes the form of a silhouette around the prototype, and usually does not effect the readability of the document, as seen in most of the residual maps of Figure 13 (an example also appears in Figure 5c). The second type results from ambiguity between two similar components. An example of this is shown in Figure 13 where the symbol 'C' was used as a prototype for the pattern 'G' in "Encore GigaMax". The third type arises when no appropriate clusters are available, as shown in Figure 13 for the letter 'E'. This case arises when the overhead of creating a cluster is higher than just tagging the component as a graphic entity and representing the entire component in the residual map. In a large document set, the number of instances of this case will be small, since each symbol will usually occur multiple times.

Due to the large size of the residual maps it is desirable to order them structurally. Since characters degrade around their edges, as suggested by Kanungo et al. [19], it is reasonable to conclude that the least significant portions of the maps are around the edges of characters, while pixels farther away from the edges contribute more significant structural information. This can be confirmed by noting the lack of structural difference between a character and its boldface counterpart, while

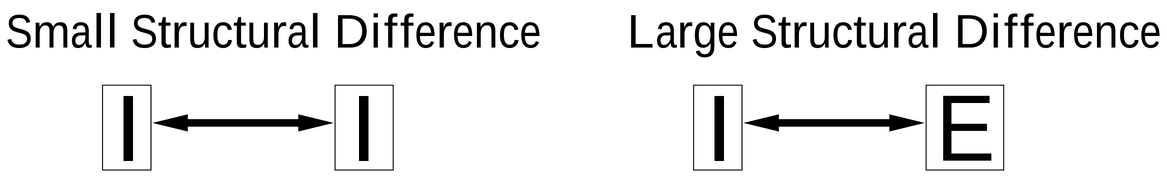


Figure 14: Comparison of structural differences of two sets of components with similar pixel differences.

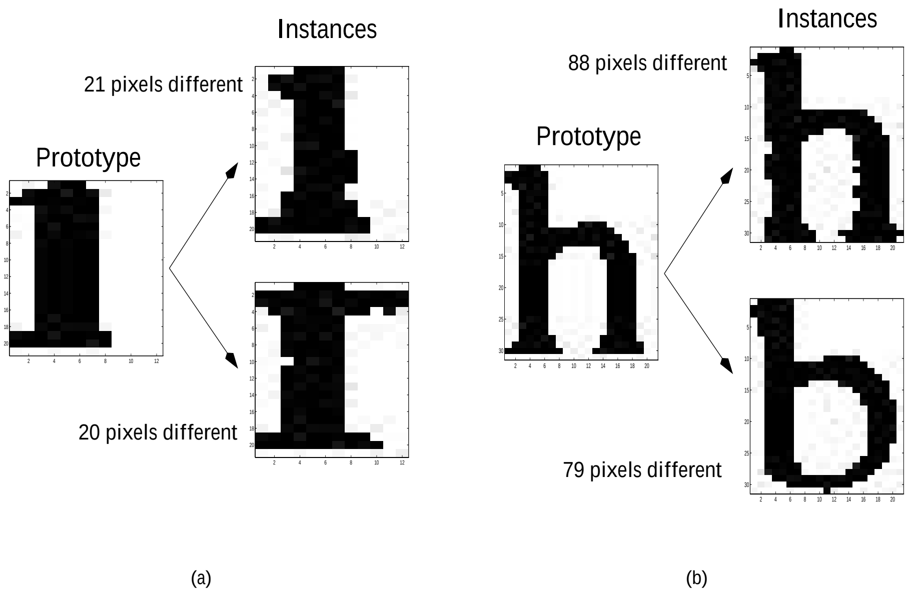


Figure 15: Examples of matching prototypes with symbol instances. A pixel difference measure may allow symbols instances with significant structural difference to be matched to a prototype.

the difference is perceivable in characters with large numbers of pixels far from the edges of the prototypes as shown in Figure 14. Figure 15 shows bitmap images in which the components, even though they have been clustered together, have significantly different structure. The out-of-class observations (Figures 15c and f) actually have lower mismatches with the prototypes (Figures 15a and b). This suggests the need for a structural component in the matching process. Structural coding can also provide a basis for lossy compression and progressive transmission. This subject is beyond the scope of this paper but has been addressed in earlier work [20, 21].

4.2 Component location specification

The specification of component location is done in two parts. For coding efficiency the page is divided into blocks of 256×256 pixels each of which can be addressed by a single byte. A component is indexed by the relative location of the upper left corner of its bounding box with respect to the upper left corner of the block. (The number of components in each block is also stored.) These locations and the block's address yield the absolute addresses of the components. This method of address specification is used in order to provide direct access to the components. Relative position specification is used in compression by Witten et al. [45], by specifying offset location from the bottom right of the previous component to the top right of the next component. Their method has smaller storage requirements but does not provide direct access to the components.

5 Page representation

In order to provide spatial access to image components in the compressed domain, it is convenient to encode four types of information in four independent sections: the file header section, the prototype section, the symbolic section (see below), and the residual section. The file header contains general information which provides indexes to the prototype, symbolic, and residual sections and an index to the header section of the subsequent page in a multi-paged document. (In this paper we have considered only single-page documents.) The prototype section is a collection of prototype bitmaps and their sizes. The symbolic section provides an encoding of the locations of components in the image and the prototype of which each component is an instance. The residual section contains the residuals produced after substituting the prototypes for the actual components. Each of these four sections is encoded as a set of streams. Figure 16 shows the organization of the streams; they are described in greater detail in the following subsections.

5.1 Streams

The file header section contains a single **header stream** and serves as an overall roadmap for the representation, specifying global document image parameters as well as indices to the prototype, symbolic, and residual sections and the streams they contain. The prototype section encodes the library of prototypes and contains two streams, the **prototype size stream**, which records the size

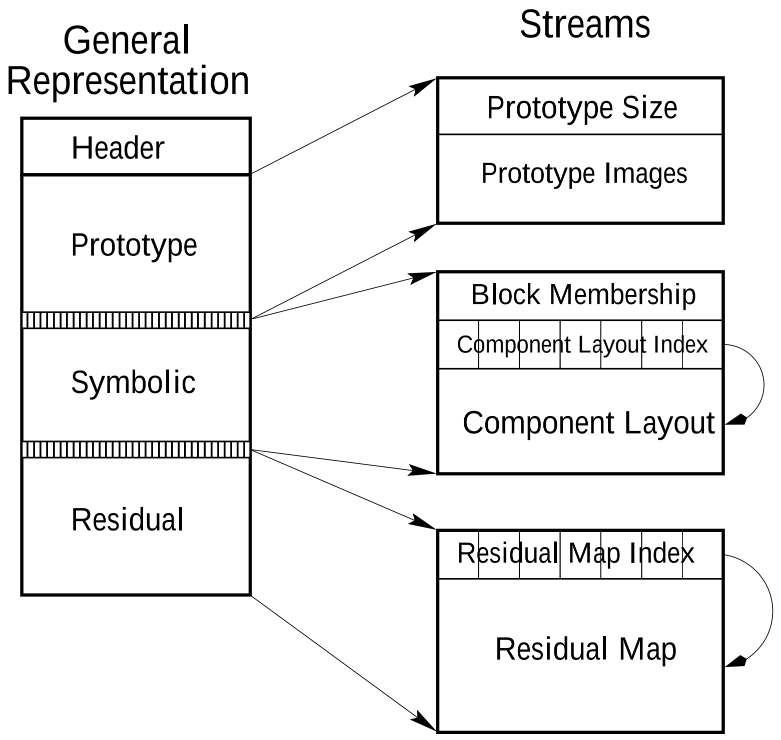


Figure 16: Data structure organization

of each prototype bitmap, and the **prototype image stream** which encodes the actual bitmaps. Two extra NULL prototypes are used to serve as indexes for small and large graphic components which are represented in the residual stream. These prototypes are used to specify appropriate numbers of bytes for the sizes of the graphic components so they are indexable.

The symbolic section specifies the location of each component along with the prototype which represents it. To reduce storage requirements in recording the component locations, the image is divided into 256×256 blocks and component locations are specified relative to each block. To calculate the absolute address of a component during decompression, the **block membership stream** provides a block index and the **component layout stream** provides the vertical and horizontal offsets of the components within the block along with their prototype labels. The **component layout index stream** is used to index the component layout stream after it is compressed.

The residual section stores the residual for each component and is indexed in the same way and in the same order as its symbolic counterpart. This requires the **residual map index stream**, an index into the residual map, and the **residual map stream** itself.

The advantage of representing different types of information by different streams and then indexing the streams is that applications which need access only to limited information do not need to decompress the entire document. For tasks such as skew estimation and correction, which may require only the locations of the components, we need access only to the prototype size and component layout streams. For a task such as keyword searching or OCR, the prototype needs to be accessed, and minimal access to the residual map stream may also be needed. To extract and decode subregions of the document image, we can access the needed blocks, and we can then access the components in these blocks efficiently using the index structures.

The size of each stream is specified in a stream header. The header remains uncompressed so that access to each stream is independent of access to the other streams. Each stream can be compressed individually or a common "dictionary" can be implemented and made available in the header. In our implementation, we index into the component layout stream by referencing block boundaries and into the residual map stream by referencing individual components.

5.2 Compressing the streams

Huffman coding is used to compress the streams; it requires two steps. First the streams are scanned to create a common Huffman lookup table. Second the individual streams are transformed according to the newly created table. Creating indexes into the compressed streams will be discussed below.

5.2.1 Header section

Header stream: This stream consists of indices to the streams contained in the prototype, symbolic, and residual sections. The indices are stored as the sizes of the individual streams; they remain uncompressed.

5.2.2 Prototype section

Prototype size stream: Prototypes are limited to 256×256 pixels so we can use single bytes for their horizontal and vertical dimensions. These pairs are ordered as a stream and compressed. The compressed size is recorded in the header. When required, the compressed stream can be decompressed and used to index the prototype image stream.

Prototype image stream: The prototype image stream orders the pixels of each prototype, either by row or column. The resulting array is packed into the prototype image stream and compressed. The compressed size is recorded in the header. For decompression, the compressed stream is indexed by the size stream. Knowing the sizes and number of prototypes from the prototype size stream, and the size of the decompressed prototype image stream, the stream can be unpacked into the set of two-dimensional prototypes.

5.2.3 Symbolic section

Block membership stream: We have divided the document image into 256×256 non-overlapping blocks. To index the blocks, we record the first row and column of the block in two bytes. We create a stream for the number of components in each block, compress the stream and record the size of the compressed stream in the header. During decompression we extract the stream and use it to index the blocks.

Component layout stream: For each component in a block, we append to a list its position in the block, by specifying a horizontal and vertical offset, and the cluster to which it belongs, by specifying the identification of its prototype. If the component is associated with a NULL prototype, we also record its size (the height and width of its bounding box). For each block, we compress the list and record its compressed size in the header. We add this size to the component layout index stream (described below) in one byte and start a new record for the next block. When all components in all blocks are exhausted, we record the overall stream size. For full decompression, we ignore the index stream and decompress the entire component layout stream. For partial decompression we use the size information in the index stream and decompress only the needed part of the component layout stream.

Component layout index stream: To index the component layout stream we form a stream based on the compressed sizes of the component layout records. We compress the entire stream and record the resulting size in the header. When the information in the stream is needed, we decompress the entire stream.

5.2.4 Residual section

Residual map stream: Using the same encoding as for the component layout stream, we order the residual maps into a stream based on their distance maps. For each residual map, we first pack the pixels into bytes, submit that portion of the stream for compression, record the compressed size in one byte, and record the ending bit offset in one byte. We do this for all residual images and record the overall size of the compressed residual map stream in the header. For full decompression, we extract the entire residual map stream based on its compressed size and unpack it. For partial decompression, we use the decompressed residual map index stream to extract the needed segments from the residual map stream.

Residual map index stream: For the index, we form a stream based on the compressed size of the list of residual map streams for each component. We compress the entire stream and record the resulting size in the header. When the information in the stream is needed, we decompress the entire stream.

6 Implementation

To implement our system the first task is to identify appropriate patterns in the image to be clustered. For Latin text, we use connected components of the binary image which are extracted by examining successive scan lines. Components on a scan line are tagged and if successive scans show connections between two components the components are merged. The contents of each component's bounding box are input to the distance-based template matching clusterer which uses a small threshold value for assignment to a cluster. If too many clusters are created, based on the number of clusters created relative to the number of objects observed, the threshold is increased. NULL prototypes provide better cluster creation by avoiding contamination from out-of-class observations. Once clustering is performed, clusters with small numbers of members are discarded and their members are tagged as graphic entities and represented as residuals. The residual map is then created for each component and ordered based on the distance transform. (The ordering of residual pixels is discussed in [21].) We pack the residual map to eight bits for better handling, and record its size as the number of bytes and left-over bits in the indexing stream. All the streams are then Huffman coded for compression by creating two library codes, one for the residual map stream and the other for the rest of the streams.

To test the system we used text regions from the first 122 scanned images (A001BIN.TIF to A006N.TIF) in the UWASH I database which is available from the University of Washington

[11]. For synthetic images we used LaTeX-generated files from the same database (L000SYN.TIF - L00MSYN.TIF). Our basic compression package converts binary TIFF images to and from symbolically compressed images. For synthetic, LaTeX-generated images, it took a total of 855 seconds to compress 23 files for an average of 37 seconds. The average resulting file size was 45312 bytes compared to 57848 bytes in CCITT G4 coding and 82112 bytes in CCITT G3. For comparison, the same images were compressed using packed bits and LZW compression, yielding average file sizes of 198,015 bytes and 110,810 bytes respectively. With original images of 2550 by 3300 pixels, the compression ratios are summarized in Table 1. For the case of scanned images, compression time was an average of 66 seconds for the 122 files of Table 1.

Image	Symbolic	MG	CCITT-G4	CCITT-G3	LZW	PACK-BITS
Synthetic	23.2	39	18.2	12.8	9.5	5.3
Scanned	13.0	27	17.8	9.7	8.7	5.5

Table 1: Compression ratios for synthetic and scanned images for different compression schemes

7 Applications in the compressed domain

This section discusses applications which can take advantage of our representation, and algorithms which implement those applications. Performance is discussed, where applicable, for algorithms running on a SparcStation 20 with the Solaris 2.5 operating system.

7.1 Lossy compression

A useful option in compressing an image is to be able to specify a quality of compression, or a required size limitation. In document image compression, it is hard to identify a measure of quality, which has to be related to the readability of the document. Most measures of quality are related to the overall "power" in the error image, defined by the difference between the lossless and lossy representations. We have found, however, that some pixels contribute more to the readability of a document than others, and even if two representations give the same "error power", one may be more readable than the other. As described previously in our representation, we have ordered the residual pixels associated with each component and can use different fractions of the

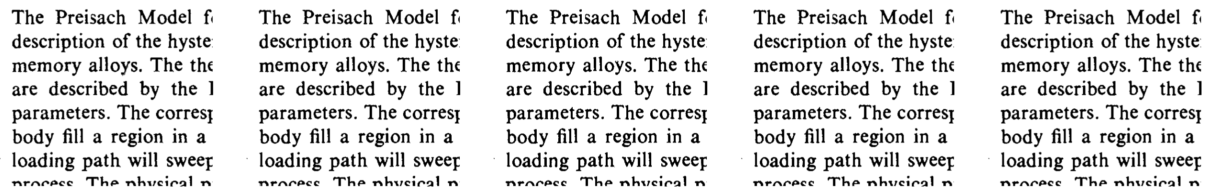


Figure 17: Lossy compression of images using a) 80%, b) 60%, c) 40%, d) 20%, and e) 1% of the error streams associated with the components.

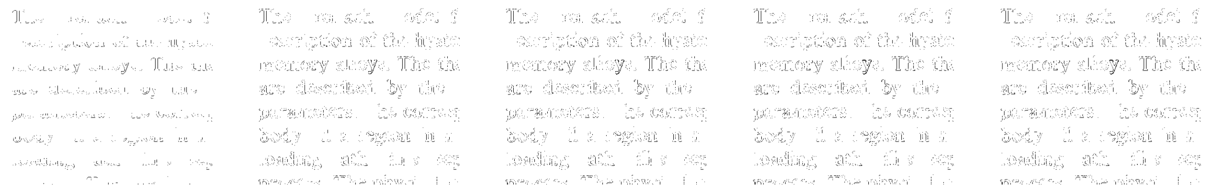


Figure 18: Difference between the original portion of the image and its lossy compression using a) 80%, b) 60%, c) 40%, d) 20%, and e) 1% of the error streams associated with the components.

residual map stream to achieve different degrees of lossy compression. We take the rendition of the document without the use of any residual pixels as a baseline and add fractions of the residual stream, increasing linearly.

Figure 17 shows an image which has been lossily compressed to varying degrees, with (a) being the least compressed and (e) being the most. The percentage indicates the amount of residual information kept for each non-NULL-clustered symbol (i.e., 100% would yield a lossless image). Note that there is no observable difference between the least and most compressed images as regards readability. The residual images shown in Figure 18 show that the residuals lie mostly on the edges of the symbols, and that even at the highest levels of lossy compression, the rendition is quite readable without any compromise in resolution. For the 122 images we achieved an average compression ratio of 29.56 for 1% of the residual stream, 25.71 for 20%, 22.88 for 40%, 19.96 for 60%, and 15.67 for 80% of the residual stream. This is in comparison to lossless compression ratios of 12.2 and 17.8 for symbolic compression and for Group 4 standard respectively.

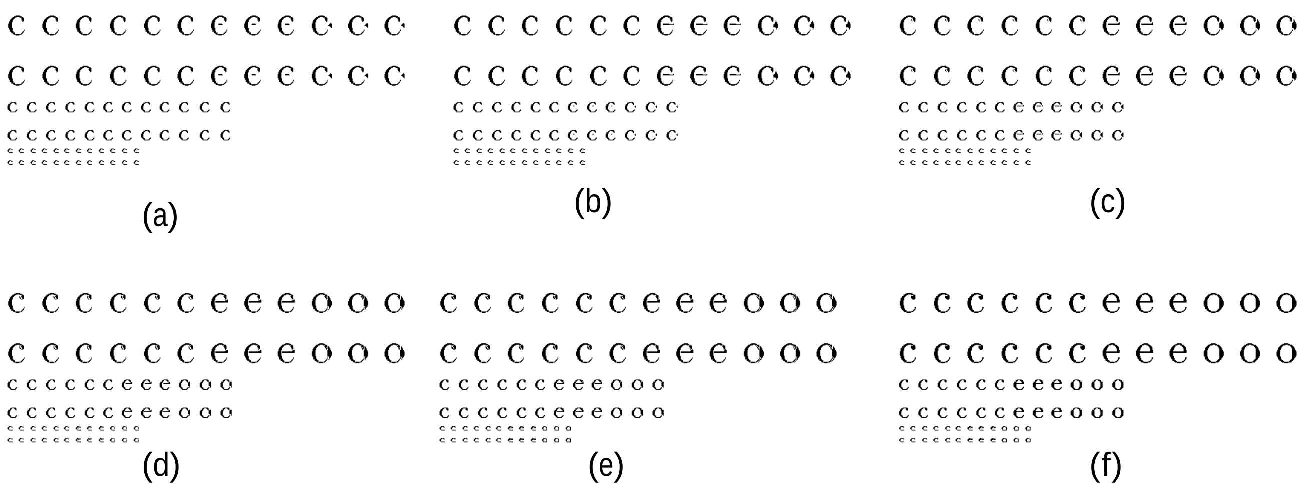
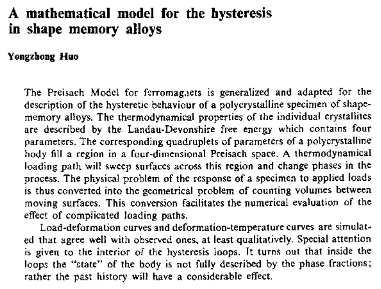


Figure 19: Progressively sending an increasing number of bits, from a) to the lossless rendition f).

7.2 Progressive transmission

In transmitting images, an ordered stream of information should be used to convey increasing amounts of information, so that the image can be rendered losslessly, if the entire stream is transmitted. In transmission of textured regions, taking a blockwise DCT (Discrete Cosine Transform) of the image and sending one coefficient per block at each iteration provides a reasonable progressive representation of the image. In the case of document images and the symbolic representation, we first transmit all the prototype and symbolic information, and we order the residual maps based on the distance transforms of the prototypes (or more precisely: of their complements). Sending the residual maps in this order, largest distance first, renders the document quite readable at the start and provides additional detail at each transmission iteration.

We have successfully implemented a progressive transmission system based on our representation. Figure 19 shows a progression for a test image which consists of instances of similar symbols; we transmit pixels that are farthest from the prototypes' edges first. Since larger components have residual pixels with larger absolute distances, they are reconstructed first, which is desirable since larger components are perceptually more salient than smaller components.



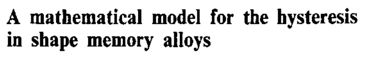
A mathematical model for the hysteresis in shape memory alloys

Yongzhong Huo

The Preisach Model for ferromagnets is generalized and adapted for the description of the hysteretic behaviour of a polycrystalline specimen of shape-memory alloys. The thermodynamical properties of the individual crystallites are described by the Landau-Devonshire free energy which contains four parameters. The corresponding quadruplets of parameters of a polycrystalline body fill a region in a four-dimensional Preisach space. A thermodynamical loading path will sweep surfaces across this region and change phases in the process. The physical problem of the response of a specimen to applied loads is thus converted into the geometrical problem of counting volumes between moving surfaces. This conversion facilitates the numerical evaluation of the effect of complicated loading paths.

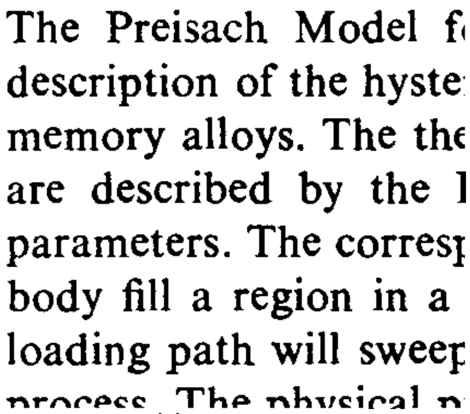
Load-deformation curves and deformation-temperature curves are simulated that agree well with observed ones, at least qualitatively. Special attention is given to the interior of the hysteresis loops. It turns out that inside the loops the "state" of the body is not fully described by the phase fractions; rather the past history will have a considerable effect.

(a)



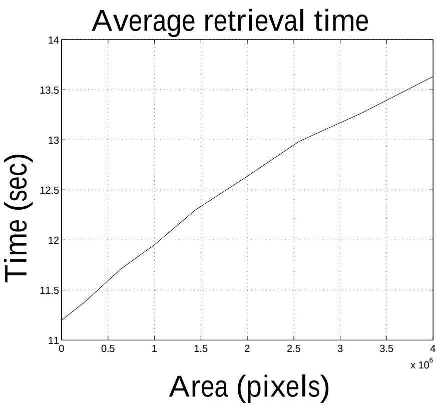
A mathematical model for the hysteresis in shape memory alloys

(b)



The Preisach Model f
description of the hyste
memory alloys. The the
are described by the l
parameters. The corresp
body fill a region in a
loading path will sweep
process. The physical p

(c)



(d)

Figure 20: Subimages of image A001BIN.TIF a) (700,700)-(2300,2000); b) (700,700)-(1900,950); c) (830,1090)-(1280,1490); d) Retrieval time as a function of area

7.3 Sub-document retrieval

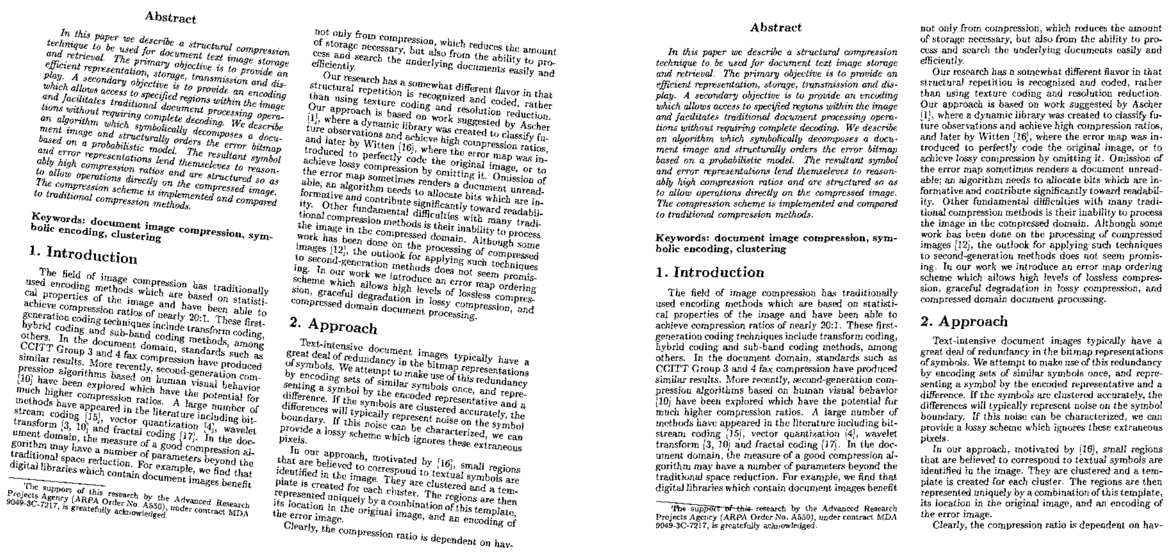
A common document browsing task requires retrieval and decoding of a subregion of the document image. This may be useful, for example, for expanding a region such as a single article or picture from a thumbnail of a compressed newspaper page. This can be done using our representation by partially decoding the appropriate streams. Specifically, we decompress the prototype size, prototype image, block membership, component layout index, and residual map index streams, and only partially decompress the largest streams, the component layout and residual map streams. Using the layout index stream we decompress components in blocks which overlap with the subimage, and according to the overlap of each component with the subimage we decompress its residual map. An example is shown in Figure 20a. If the coordinates of a subtitle are known, we may decompress only the subtitle (Figure 20b), and if the top left corner of the first paragraph is needed, we can decompress accordingly (Figure 20c). We can decompress only the top part of the first page in a document to determine the title and authors of the document. The processing time depends on the area of the region of interest. Figure 20d shows a plot of decompression time versus retrieval area. The time required to decompress small regions can be viewed as overhead time, and the increase in time with area defines the scalability.

7.4 Skew estimation and correction

Skew estimation and correction are also important tasks in an OCR system. By using an algorithm based on the Hough transform, we are able to estimate skew and correct it using our representation, without having to fully decompress. We use only the position and size of each component. To compute their values, we decompress the prototype size, block membership, and component layout streams. We input the coordinates of the middle of the bottom of each component to a Hough transform, and thus compute the skew. For the 122 images in our database, it took an average of 2.5 seconds and 9152 bytes per image to calculate skew to an accuracy of 1/640 vertical units per horizontal unit. On the same set of test documents the average error was measured to be 0.1786 degrees. For skew correction, we modified the component layout and residual map streams and reordered them if the components moved from one block to another. Reordering is computationally expensive and does not contribute much correction for small skew angles. To improve performance, we bounded the movement of the components across a block boundary to five pixels; below this bound, movement is terminated at the edge of the block. For instance, if a component is located close to a block boundary, and by deskewing, it moves two pixels into another block, it is moved to the edge of its original block. We take the middle of the image as the reference point. The translation in the vertical direction is computed by multiplying the horizontal distance from the middle of the page by the slope of the line containing the reference point and component location. Figure 21 shows two examples of skewed documents and the results of deskewing them. For the 122 test images, it took an average of 3.3 seconds to deskew an image.

7.5 Keyword Search

Keyword search is another important document-related task. A method reported by Spitz [39] requires scan-line ordering of components and matching of ordered components to queried components. To implement this method using our representation, we decompress the prototype size, prototype image, block membership and component layout streams. The skew angle of the document is first estimated, as described above, and the bounding boxes are horizontally projected. The projection is then scanned to determine the locations of lines of text. We scan the lines from top to bottom and record the components which each line contains. This method picks up disjoint components, such as the dots in the 'i' and 'j', very nicely. It is also very stable to errors in locations



Abstract

In this paper we describe a structural compression technique to be used for document text image storage and retrieval. The primary objective is to provide an efficient representation, storage, transmission and display. A secondary objective is to provide an encoding which allows access to specified regions within the image and facilitates traditional document processing operations without requiring complete decoding. We describe an algorithm which symbolically decomposes a document image and structurally orders the error bitmap based on a probabilistic model. The resultant symbol and error representations lend themseleves to reasonably high compression ratios and are structured so as to allow operations directly on the compressed image. The compression scheme is implemented and compared to traditional compression methods.

Keywords: document image compression, symbolic encoding, clustering

1. Introduction

The field of image compression has traditionally used encoding methods which are based on statistical properties of the image and have been able to achieve compression ratios of nearly 20:1. These first-generation coding techniques include transform coding, hybrid coding and sub-band coding methods, among others. In the document domain, standards such as CCITT Group 3 and 4 fax compression have produced similar results. More recently, second-generation compression algorithms based on human visual behavior [10] have been explored which have the potential for much higher compression ratios. A large number of methods have appeared in the literature including bit-stream coding [15], vector quantization [4], wavelet transform [3, 10] and fractal coding [17]. In the document domain, the measure of a good compression algorithm may have a number of parameters beyond the traditional space reduction. For example, we find that digital libraries which contain document images benefit not only from compression, which reduces the amount of storage necessary, but also from the ability to process and search the underlying documents easily and efficiently.

Our research has a somewhat different flavor in that structural repetition is recognized and coded, rather than using texture coding and resolution reduction. Our approach is based on work suggested by Ascher [1], where a dynamic library was created to classify future observations and achieve high compression ratios, and later by Witten [16], where the error map was introduced to perfectly code the original image, or to achieve lossy compression by omitting it. Omission of the error map sometimes renders a document unreadable; an algorithm needs to allocate bits which are informative and contribute significantly toward readability. Other fundamental difficulties with many traditional compression methods is their inability to process the image in the compressed domain. Although some work has been done on the processing of compressed images [12], the outlook for applying such techniques to second-generation methods does not seem promising. In our work we introduce an error map ordering scheme which allows high levels of lossless compression, graceful degradation in lossy compression, and compressed domain document processing.

2. Approach

Text-intensive document images typically have a great deal of redundancy in the bitmap representations of symbols. We attempt to make use of this redundancy by encoding sets of similar symbols once, and representing a symbol by the encoded representative and a difference. If the symbols are clustered accurately, the differences will typically represent noise on the symbol boundary. If this noise can be characterized, we can provide a lossy scheme which ignores these extraneous pixels.

In our approach, motivated by [16], small regions that are believed to correspond to textual symbols are identified in the image. They are clustered and a template is created for each cluster. The regions are then represented uniquely by a combination of this template, its location in the original image, and an encoding of the error image.

Clearly, the compression ratio is dependent on hav-

~~The support of this~~ research by the Advanced Research Projects Agency (ARPA Order No. A550), under contract MDA 9049-3C-7217, is greatefully acknowledged.

Figure 21: Deskewing example: a) Skew at five degrees, b) five degree deskew

of objects and in scan direction.

Once the components are ordered we apply a feature-based matching algorithm. We classify each component as being an x-height, an ascender, a descender, an "i", a "j", or punctuation. A second-level classification checks if there exists a hole in the component, and a third checks for the existence of a concavity from the right. These features are computed only for the prototypes rather than for the individual components. The heights of the components are measured only for the first ten instances of each prototype in the image, and their values are averaged. The input query is mapped into this feature space using a simple lookup table; 90% of the maximum score is taken to be a match. Figure 22 shows search results on the 122 database images for the query "approach". It took an average of 2.7 seconds per image to obtain the results. There are fundamental ambiguities associated with using our small set of features; for example, we are treating characters like "a" and "o" as belonging to the same feature class. However, the effects of these ambiguities are greatly reduced when a string of features is used in matching.

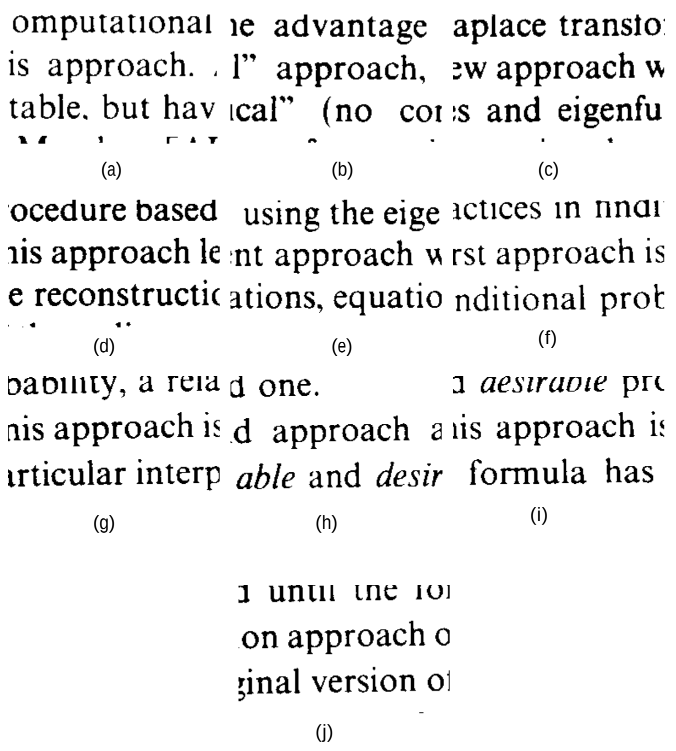

(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)
(i)
(j)

Figure 22: Query results for the string "approach" on selected images from the University of Washington Database [11].

8 Conclusion

Computer systems are being increasingly used to process and analyze images of many forms. These tasks vary in computational complexity and in storage and communication requirements. Advances in data compression have been able to reduce these requirements; a compressed file takes less storage space, takes less time to read and process, and takes less channel capacity to transmit. Compression is especially important in handling document images due to the large file sizes involved and the large amount of processing they require. With the increasing popularity of digital libraries, retaining and presenting scanned documents in image form remains a cost-effective and usually accurate way to distribute document information that was originally in hard copy.

We have developed a system that addresses all aspects of document image compression, especially those related to transmission and processing. Our system achieves compression while still allowing for efficient processing. It is evident that the important information in documents is at the symbol level, and the structure of the symbols needs to be preserved. This implies that we cannot use traditional resolution-reduction methods to achieve compression. Instead, we take a pattern-matching and substitution approach. In this approach, we represent information hierarchically, and we create a residual coding that is structurally efficient. It remains only to demonstrate processing efficiency. This paper has described methods of performing a wide variety of document processing tasks. We have shown that symbolic coding is very efficient for image retrieval. Image analysis tasks such as skew estimation, correction, and keyword searching can also be performed efficiently since the symbolic representation preserves sufficient information to be used in these tasks.

In this paper we have addressed the problem of compression strictly in the image domain, and for good reason. It is clear that the ultimate form of symbolic compression is simply to recognize the symbols and represent them by their ASCII codes. However, such a compressed representation suffers by not preserving information about the fonts, point sizes, locations on the page, etc. To varying degrees such features can be preserved, but the representation does not allow us to reconstruct a truly lossless version of the original. A hybrid compression scheme which integrates ASCII symbols which can be recognized with high accuracy into a scheme such as ours would be ideal.

References

[1] M.R. Anderberg. *Cluster Analysis for Applications.* Academic Press, 1973.

[2] R. Ascher and G. Nagy. A means for achieving a high degree of compaction on scan-digitized printed text. *IEEE Transactions on Computers*, 23:1174–1179, 1974.

[3] M. Atallah, Y. Genin, and W. Szpakowski. Pattern matching image compression: Algorithmic and empirical results. Technical Report CSD TR-95-083, Computer Science Department, Purdue University, 1995.

[4] H.S. Baird. The skew angle of printed documents. In *Proceedings of the SPSE 40th Annual Conference and Symposium on Hybrid Imaging Systems*, pages 21–24, 1987.

[5] M.F. Barnsley and L.P. Hurd. *Fractal Image Compression.* A.K. Peters, 1993.

[6] G. Bessho, K. Ejiri, and J.F. Cullen. Fast and accurate skew detection algorithm for a text document or a document with straight lines. In *Proceedings of the SPIE - Document Recognition*, volume 2181, pages 133–140, 1994.

[7] D. Bodson, S. Urban, A. Deutermann, and C. Clarke. Measurement of data compression in advanced group 4 facsimile system. *Proceedings of the IEEE*, 73:731–739, 1985.

[8] N. Faller. An adaptive system for data compression. In *Proceedings of the Asilomar Conference on Circuits, Systems and Computers*, pages 593–597, 1973.

[9] R. Gallager. Variations on a theme by Huffman. *IEEE Transactions on Information Theory*, 24:668–674, 1978.

[10] A. Gersho and R. Gray. *Vector Quantization and Signal Compression.* Kluwer Academic Publishers, Boston, 1992.

[11] R. Haralick. UW English document image database I: A database of document images for OCR research. CDROM.

[12] M. Holt and C. Xydeas. Recent developments in image data compression for digital facsimile. *ICL Technical Journal*, pages 123–146, 1986.

[13] P. Howard. Lossless and lossy compression of text images by soft pattern matching. In *Proceedings of the IEEE Data Compression Conference*, pages 210–219, 1996.

[14] T. Huang. Run length coding and its extensions. In T. Huang and O. Tretiak, editors, *Picture Bandwidth Compression*, pages 231–264. Gordon and Breach, 1972.

[15] K. Culick II and V. Valenta. Finite automata based compression of bi-level images. In *Proceedings of the IEEE Data Compression Conference*, pages 280–289, 1996.

[16] S. Inglis and I. Witten. Compression-based template matching. In *Proceedings of the IEEE Data Compression Conference*, 1994.

[17] A. Jain. *Fundamentals of Digital Image Processing.* Prentice-Hall, 1989.

[18] O. Johnsen, J. Segen, and G. Cash. Coding of two-level pictures by pattern matching and substitution. *Bell System Technical Journal*, 62:2513–2545, 1983.

[19] T. Kanungo, R.M. Haralick, and I.T. Phillips. Global and local document degradation models. In *Proceedings of the International Conference on Document Analysis and Recognition*, pages 730–734, 1993.

[20] O. Kia and D. Doermann. Structure-preserving image compression and transmission. In *Proceedings of the International Conference on Image Processing*, volume I, pages 193–196, 1996.

[21] O. Kia and D. Doermann. Symbolic compression for document analysis. In *Proceedings of the International Conference on Pattern Recognition*, volume III, pages 664–668, 1996.

[22] O. Kia, D. Doermann, and R. Chellappa. Compressed-domain document retrieval and analysis. In *Proceedings of the SPIE - Multimedia Storage and Archiving Systems*, volume 2916, 1996.

[23] D. Knuth. Optimal binary search trees. *Acta Informatica*, 1:14–25, 1971.

[24] J. Liu, C.M. Lee, and R.B. Shu. An efficient method for the skew normalization of a document image. In *Proceedings of the International Conference on Pattern Recognition*, volume III, pages 122–125, 1992.

[25] T. Luczak and W. Szpakowski. A lossy data compression based on an approximate pattern matching. Technical Report CSD TR-94-072, Computer Science Department, Purdue University, 1995.

[26] C. Maa. Identifying the existence of bar codes in compressed images. *CVGIP: Graphical Models and Image Processing*, 56:352–356, 1994.

[27] K. Mohiuddin. *Pattern Matching with Application to Binary Image Compression.* PhD thesis, Stanford University, 1982.

[28] K. Mohiuddin, J. Rissanen, and R. Arps. Lossless binary image compression based on pattern matching. In *Proceedings of the International Conference on Computers, Systems, and Signal Processing*, pages 447–451, 1984.

[29] S. Mori, C.Y. Suen, and K. Yamamoto. Historical review of OCR research and development. *Proceedings of the IEEE*, 80:1029–1058, 1992.

[30] Y. Nakano, Y. Shima, H. Fujisawa, J. Higashino, and M. Fujinawa. An algorithm for the skew normalization of document image. In *Proceedings of the International Conference on Pattern Recognition*, pages 8–13, 1990.

[31] L. O'Gorman. The document spectrum for page layout analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15:1162–1173, 1993.

[32] T. Pavlidis and J. Zhou. Page segmentation and classification. *CVGIP: Graphical Models and Image Processing*, 54:484–496, 1992.

[33] W. Pennebaker and J. Mitchell. Probability estimation for the Q-Coder. *IBM Journal of Research and Development*, 32:737–752, 1988.

[34] W. Pennebaker, J. Mitchell, G. Langdon, and R. Arps. An overview of the basic principles of the Q-Coder adaptive binary arithmetic coder. *IBM Journal of Research and Development*, 32:717–726, 1988.

[35] W. Postl. Detection of linear oblique structures and skew scan in digitized documents. In *Proceedings of the International Conference on Pattern Recognition*, pages 687–689, 1986.

[36] W. Pratt, P. Capitant, W. Chen, E. Hamilton, and R. Willis. Combining symbol matching facsimile data compression system. *Proceedings of the IEEE*, 68:786–796, 1980.

[37] K. Rose. *Deterministic Annealing, Clustering and Optimization.* PhD thesis, California Institute of Technology, 1991.

[38] C. Shannon. A mathematical theory of communication. *Bell Systems Technical Journal*, 27:623–656, 1948.

[39] A. Spitz. An OCR based on character shape codes and lexical information. In *Proceedings of the International Conference on Document Analysis and Recognition*, pages 723–728, 1995.

[40] A.L. Spitz. Skew determination in CCITT Group 4 compressed document images. In *Proceedings of the First Symposium on Document Analysis and Information Retrieval*, pages 11–25, 1992.

[41] A.L. Spitz. Logotype detection in compressed images using alignment signatures. In *Proceedings of the Fifth Symposium on Document Analysis and Information Retrieval*, pages 303–310, 1996.

[42] J. Vitter. Design and analysis of dynamic Huffman codes. *Journal of the Association for Computing Machinary*, 34:825–845, 1987.

[43] I. Witten, T. Bell, H. Emberson, S. Inglis, and A. Moffat. Textual image compression: Two-stage lossy/lossless encoding of textual images. *Proceedings of the IEEE*, 82:878–888, 1994.

[44] I. Witten, T. Bell, M. Harrison, M. James, and A. Moffat. Textual image compression. In *Proceedings of the IEEE Data Compression Conference*, pages 42–51, 1992.

[45] I. Witten, A. Moffat, and T. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Van Nostrand Reinhold, 1994.

[46] K. Wong, R. Casey, and F. Wahl. Document analysis system. *IBM Journal of Research and Development*, 26:647–656, 1982.

[47] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23:337–343, 1977.

[48] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24:530–536, 1978.